Outlier Removal in 2D Leap Frog Algorithm

Ryszard Kozera[1,2] and Jacek Tchórzewski

[1] Faculty of Mathematics and Information Science
Warsaw University of Technology
00-661 Pl. Politechniki 1, Warsaw, Poland
[2] Faculty of Applied Informatics and Mathematics
Warsaw University of Life Sciences - SGGW
02-776 Nowoursynowska 159, Warsaw, Poland
r.kozera@mini.pw.edu.pl, tchorzewski.jacek@gmail.com

Abstract. In this paper a 2D Leap Frog Algorithm is applied to solve the so-called *noisy Photometric Stereo* problem. In 3-source Photometric Stereo (noiseless or noisy) an ideal unknown Lambertian surface is illuminated from distant light-source directions (their directions are assumed to be linearly independent). The subsequent goal, given three images is to reconstruct the illuminated object's shape. Ultimately, in the presence of noise, this problem leads to a highly non-linear optimization task with the corresponding cost function having a large number of independent variables. One method to solve it is *2D Leap Frog Algorithm.* During reconstruction, problem that commonly arises, renders the *outliers* generated in the retrieved shape. In this paper we implement 2D Leap Frog. In particular we focus on choosing snapshot size and on invoking two algorithms that can remove outliers from reconstructed shape. Performance of extended 2D Leap Frog is illustrated by examples chosen especially to demonstrate how this solution is applicable in computer vision. Remarkably, this optimization scheme can also be used for an arbitrary optimization problem depending on large number of variables.

Keywords: Shape from Shading, noise removal, optimization, outliers.

1 Introduction

One of the most important problems in the field of computer vision is an issue of shape reconstruction from its image(s) data. There are two main approaches to tackle this problem. One method (discussed in this article) is based on surface S reconstruction from the data obtained from its image(s) created by single camera (the so-called *Shape from Shading* [4]). The second class of techniques is the shape reconstruction based on multiple camera input data and the usage of *triangulation like methods* [5]. In the first method single illumination yields the so-called single image Shape-from-Shading problem. On the other hand a multiple illumination in Shape from Shading is called Photometric Stereo (see e.g. Horn [3], Kozera [8] and Noakes and Kozera [9]). As opposed to the ideal continuous setting in Shape from Shading (see e.g. [4]) an additional problem arises when noise is added to the image(s). This forms noisy Photometric Stereo problem (or noisy single image Shape from Shading problem, respectively). As it turns out it is modeled by the corresponding highly non-linear optimization task in many multiple variables. One of the computational techniques striving to deal with such optimization is a 2D Leap Frog Algorithm [6] and [9], that is recalled and modified in this paper.

While dealing with classical Shape from Shading, one seeks a function $u : \Omega \subseteq \mathbb{R}^2 \mapsto \mathbb{R}$, that represents distance of surface point $(x, y, u(x, y)) \in graph(u) = S$ from a camera (where Ω is a picture within the camera). For a Lambertian surface (which is a perfect diffuser) with a constant albedo, illuminated from distant light-source direction $p = (p_1, p_2, p_3)$ the image irradiance equation in the *continuous setting* is given over Ω as (see Horn [3]):

$$E_p(x,y) = \frac{p_1 u_x(x,y) + p_2 u_y(x,y) - p_3}{\sqrt{p_1^2+p_2^2+p_3^2}\sqrt{u_x^2(x,y)+u_y^2(x,y)+1}}. \tag{1}$$

For the remaining two light-source directions $q = (q_1, q_2, q_3)$ and $r = (r_1, r_2, r_3)$, similar image irradiance equations can be derived. Note that we urge $det(p, q, r) \neq 0$ (i.e. light-source directions are to be linearly independent).

In the case of *discrete model* (e.g. with noise added) a pertinent function is to be minimized, later called a cost function $\mathcal{E}$. It is derived from the physical concepts of Photometric Stereo and from the properties of a Lambertian surface. In real case our Ω is not continuous but discrete as it addresses the image(s) represented by a collection of pixels. Assume that the number of all image pixels is n^2. The general formula for such cost function (by (1)) reads (see Noakes and Kozera [9]):

$$\mathcal{E}(u) = \sum_{i,j=2}^{n-1} \mathcal{E}_{i,j}(u), \tag{2}$$

where $u \in \mathbb{R}^{n^2-4}$ and $(i;j)$-pixel energy value $\mathcal{E}_{i,j} = \mathcal{E}_{i,j}^p + \mathcal{E}_{i,j}^r + \mathcal{E}_{i,j}^q$ with:

$$\mathcal{E}_{i,j}^p(u) = \left(\frac{p_1\left(\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}\right) + p_2\left(\frac{u_{i,j+1}-u_{i,j-1}}{2\Delta y}\right) - p_3}{\|p\|\sqrt{\left(\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}\right)^2 + \left(\frac{u_{i,j+1}-u_{i,j-1}}{2\Delta y}\right)^2 + 1}} - E_p(x_i, y_j) \right)^2, \tag{3}$$

where $\|p\| = \sqrt{p_1^2+p_2^2+p_3^2}$, and $\mathcal{E}_{i,j}^r$ and $\mathcal{E}_{i,j}^q$ are defined similarly to (3). Note that (3) is a discretized version of image irradiance equation (1) at internal image pixel point (i, j) upon using central difference derivative approximations:

$$u_x(i,j) \approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x}$$

and $u_y(x, y)$ is estimated similarly.

We look for discrete u which minimizes (2). At this moment it is clearly visible that there exists strong *nonlinearity* in formula (2). The cost function $\mathcal{E}$ depends on $n^2 - 4$ variables (image corners do not participate in reconstruction) which represents almost entire image resolution (in practice a very big number). Thus the problem (2) usually forms an optimization task depending on a very large number of variables. Using here Newton's method based on inversion $D^2\mathcal{E}$ over entire Ω is a huge computational task.

2 2D Leap Frog Algorithm

The main idea standing behind the 2D Leap Frog Algorithm is to find the sub-optimal minima of a function $\mathcal{E}$ (see (2)) by accumulating the results of the so-called local area optimizers. It is worth mentioning the difference between the terms *local area minimum* versus *local minimum* of $\mathcal{E}$. The difference is that the local area minimum is a suboptimal minimum for a cost function $\mathcal{E}$ considered with the values u of over pixels taken from some local area $\Omega_{loc} \subseteq \Omega$. On the other hand the local minimum of $\mathcal{E}$ is any sub-optimal solution to (2) over entire image Ω. Note that, according to formula (1) a visible part of the Lambertian surface corresponds to the case, where $0 \leq \cos\Theta \leq 1$, whereas the invisible one results when $-1 \leq \cos\Theta < 0$ (where Θ denotes the angle between the light-source direction and the surface normal at point $(x, y, u(x, y))$). In both cases, however the image irradiance equations can still be mathematically posed and solved (this also reffers to (2)). Hence the corresponding optimization problem (2) can be tackled at least theoretically over entire Ω, which is rectangular (camera sensor). Thus for simplicity we assume and solve the optimization problem (2) under the constraint that the data over entire rectangular image are accessible. Let us denote an image space $\Omega = [0,1] \times [0,1]$ with given $(k; l) \in \mathbb{N}$ as length and height in pixels. The space Ω is now divided into atomic subspaces, described as follows:

$$S^l_{i_q j_q} = \left|\frac{i_q - 1}{2^l}, \frac{i_q}{2^l}\right| \times \left|\frac{j_q - 1}{2^l}, \frac{j_q}{2^l}\right|, \qquad \text{for } 1 \leq i_q, j_q \leq 2^l.$$

The main part of the 2D Leap Frog Algorithm works only on these rectangular subspaces, that are each denoted by SQ^{lm}_{ij}, where i and j are the row and column indices, and l and m are the width and height of the subspace.
Assume now that $u^{k_1}_1, \ldots, u^{k_1}_s$ represents some free variables of $\mathcal{E}$ and the remaining $u^{k_2}_1, \ldots, u^{k_2}_t$ are temporarily frozen (where $s, t \in \mathbb{N}$). Then our global energy $\mathcal{E}$ can be split into two components:

$$\mathcal{E}(u) = \mathcal{E}_1\left(u^{k_1}_1, \ldots, u^{k_1}_s\right) + \mathcal{E}_2\left(u^{k_2}_1, \ldots, u^{k_2}_t\right). \tag{4}$$

Evidently minimizing $\mathcal{E}_1$ over variables $u^{k_1}_1, \ldots, u^{k_1}_s$ only, does not change the value of $\mathcal{E}_2$ and therefore the whole value of $\mathcal{E}$ is also decreased. If s is small then optimizing $\mathcal{E}_1$ becomes a computationally feasible task (e.g. for Newton's method). This is the main idea standing behind the 2D Leap Frog. Optimizing

$\mathcal{E}_1$ in our case corresponds to the optimization of $\mathcal{E}$ over each SQ_{ij}^{lm} with all other values of u frozen. As shown by Noakes and Kozera [9] an iterative sequence of local area optimizations over different overlapping snapshot yields a suboptimal solution of $\mathcal{E}$ over the entire image Ω.

2.1 2D Leap Frog Local Optimizer

There are nine different types of subspaces, also called grids (or *snapshots*), as shown in Fig. 1. These subspaces are called: *top-left(1), top(2), top-right(3), middle-left(4), middle(5), middle-right(6), bottom-left(7), bottom(8),* and *bottom-right(9),* respectively. Note that for synthetic images we can assume (as explained before) that image is rectangular. Otherwise for the real images (with shadows) nine cases have to be combined with the visible regions of images (left for future work). There are five types of pixels in each grid as follows:

1. *Type* 0 – pixels that are only used to preserve the rectangular shape of a grid, and not used for computation.
2. *Type 1* – pixels that are locked, and have constant value throughout the computation of the grid. These pixels are also excluded from the energy cost function computation. They are used to compute the central-difference derivative approximation for some pixels around them.
3. *Type 2* – these pixels are also locked, and are included in the energy function.
4. *Type 3* – these pixels are not locked, and are used as variables while searching for the minimum of the energy cost function.They are also included in the energy cost function.
5. *Type 4* – these pixels are neither locked nor included in the energy function.

The nine grid types can be further reduced to three main types: side grids (touching only 1 border), corner grids (touching 2 borders excluding top-left case) and mid grids (touching none of the borders). This is because grids of the

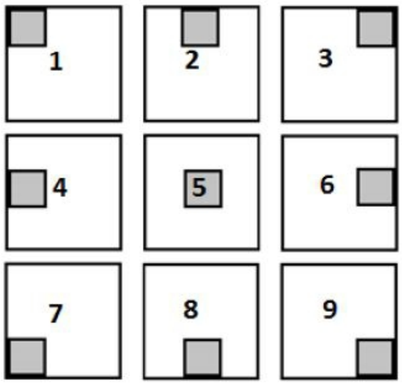
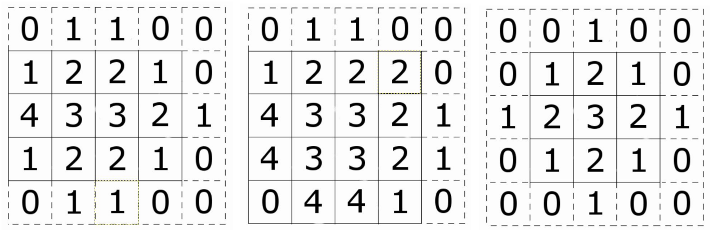


Fig. 1. All 9 types of snapshots

same type can be obtained from each other by rotation. Firstly, side grids are described as shown in Fig. 2.

The central pixels of the border type grid are of type 3. Type 4 pixels are found directly between the pixels of type 3 and the border. This is because the border pixels must be included in the overall energy function, but it is impossible to approximate their central difference derivatives, and therefore their energies cannot be computed. Type 3 pixels are also surrounded by pixels of type 2 because minimizing the cost function has an impact on their surrounding pixels due to their derivative approximation. Lastly, there are type 1 pixels surrounding all others pixels. They are used just to evaluate the derivatives of other pixels and have no further impact on the algorithm. Corner grids are localized next to two borders as shown in Fig. 2. Type 3 pixels are again in the middle of the grid with type 4 pixels situated between them and the borders. Type 2 pixels also surround type 3 pixels, and type 1 pixels surround everything. The only pixel worth mentioning is the corner pixel. This pixel needs to be of type 0 because there are no surrounding dependent pixels. This implies that the strict corners of the space never changes and remains at the initial guess.

Mid grids are the simplest cases as shown in Fig. 2. Type 4 pixels are not used because energy of all pixels in the grid can be calculated. There is a rectangle of type 3 pixels in the middle surrounded by type 2 and type 1 pixels. This case is straightforward and does not require any further explanation.



0	1	1	0	0
1	2	2	1	0
4	3	3	2	1
1	2	2	1	0
0	1	1	0	0

(a)

0	1	1	0	0
1	2	2	2	0
4	3	3	2	1
4	3	3	2	1
0	4	4	1	0

(b)

0	0	1	0	0
0	1	2	1	0
1	2	3	2	1
0	1	2	1	0
0	0	1	0	0

(c)

Fig. 2. Main three types of grids

2.2 2D Leap Frog Algorithm

The non-linear 2D Leap Frog Algorithm is constructed as follows:

1. Obtain an initial guess u_0 (e.g. using Lawn Mowing Algorithm [10]). For the need of our paper we added a large noise to the ideal continuous solution u.
2. Divide the space into grids of fixed size, overlapping in such a way that 3 columns and rows of pixels are common for grids that are neighbors.
3. Start first iteration of the algorithm with the grid on the left-top corner, and apply a minimization algorithm to obtain new values of the unlocked variables.
4. Move to the right neighbor of the previous grid. Apply minimization algorithm to the function created by the top grid case. Repeat this process until the right border is reached.

5. In the last grid of the current row, the right-top grid case is used for optimization.
6. Move to the row below. The first grid uses the left grid case for optimization.
7. Move to right neighboring grid and minimize using the mid grid case. Repeat until the right boundary is reached.
8. The last grid in this row is minimized using the right grid case.
9. Repeat 3 previous steps of the algorithm for all rows except the bottom one.
10. First grid of the last row uses the bottom-left grid case to minimize $\mathcal{E}$.
11. All consecutive grids except of the right most grid use the bottom grid case.
12. The last grid uses the right-bottom grid case.
13. Repeat all previous steps until one of the stopping condition is fulfilled.

In most cases the steps differ only in the type of grid cases that are used. 2D Leap Frog requires that, in every iteration, all pixels (except for the corner pixels) are unlocked at least once. Otherwise the algorithm does not work properly as some variables of the cost function $\mathcal{E}$ never change. In addition this guarantees that Leap Frog converges to critical point of $\mathcal{E}$.

There are *three possible stopping criteria* for the 2D Leap Frog Algorithm:

1. *Timeout*: the algorithm computation stops upon exceeding a priori established time limit.
2. *Maximum iterations limit*: the algorithm has a cap on the number of iterations that is computed.
3. *Epsilon limit*: the global cost function is not decreasing anymore; this is the most difficult criterion possible in the algorithm because the iterations can have uneven decrease. For example there can be a slowdown in decrease for a few iterations in the middle of computation; therefore the best way of measuring the decrease is for at least four iterations. In this case if the decrease is close to zero for the chosen number of iterations, the algorithm stops.

2.3 Ambiguities and Relaxation

There are two kinds of ambiguities in our discrete settings namely, *standard* and *strong ambiguities.* We explain now the difference. Recall that three images are obtained from three linearly independent light-sources. With these data a system of three image irradiance equations is generated, and independent Gaussian noise is added to the corresponding images. Assume now that C denotes the table of constant entries c: which can be obtained by deleting corner values from the matrix of dimention $n \times n$. Clearly we have:

$$\mathcal{E}\left(u + C\right) = \mathcal{E}\left(u\right).$$

Therefore adding a constant value to the reconstructed surface in the discrete problem does not change the energy $\mathcal{E}$. This is called a *standard ambiguity in discrete case.* We demonstrate now, the so-called *strong ambiguity* which also arises in the discretization of u. To see it let us take tables shown in Fig. 3.When adding these tables $u_i(i = 1, 2, 3, 4)$ to the reflectance maps, due to the central difference method, we also have:

$$\mathcal{E}\left(u + \sum_{1\le k\le 4} \breve{c}_k * u_k\right) = \mathcal{E}\left(u\right).$$

This phenomenon is called a *strong ambiguity*. Such scenario can be described in terms of finding the minima of a function of four variables. If the function u is shaped like a valley, with the lowest place on the same plane, it is impossible to find only one minimum, as the function has then same value along the valley. This creates the problem of having infinitely many solutions that have the same energy. The simplest solution is to freeze four pixels in positions: $(1; 2); (1; 3); (2; 2); (2; 3)$. This guarantees that the central differences are uniquely determined (for proof see Noakes and Kozera [9]). When the algorithm finishes its work, these four pixels need relaxation, meaning that their minimum is found. It is important to add that, Leap Frog converges to sub-global solution (if initial noise is small then it converges to the global solution which is close to correct solution). For further details refer to Noakes and Kozera [9].

$$u_1 = \begin{pmatrix} 1\;0\;1\;0 \ldots 0\;1\;0 \\ 0\;0\;0\;0\;0 \ldots 0\;0\;0\;0 \\ 0\;1\;0\;1\;0 \ldots 0\;1\;0\;1 \\ \vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots \\ 0\;0\;0\;0\;0 \ldots 0\;0\;0\;0 \\ 0\;1\;0\;1\;0 \ldots 0\;1\;0\;1 \\ 0\;0\;0\;0 \ldots 0\;0\;0 \end{pmatrix} \qquad u_2 = \begin{pmatrix} 0\;0\;0\;0 \ldots 0\;0\;0 \\ 1\;0\;1\;0\;1 \ldots 1\;0\;1\;0 \\ 0\;0\;0\;0\;0 \ldots 0\;0\;0\;0 \\ \vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots \\ 1\;0\;1\;0\;1 \ldots 1\;0\;1\;0 \\ 0\;0\;0\;0\;0 \ldots 0\;0\;0\;0 \\ 0\;1\;0\;1 \ldots 1\;0\;1 \end{pmatrix}$$

$$u_3 = \begin{pmatrix} 0\;0\;0\;0 \ldots 0\;0\;0 \\ 0\;1\;0\;1\;0 \ldots 0\;1\;0\;1 \\ 0\;0\;0\;0\;0 \ldots 0\;0\;0\;0 \\ \vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots \\ 0\;1\;0\;1\;0 \ldots 0\;1\;0\;1 \\ 0\;0\;0\;0\;0 \ldots 0\;0\;0\;0 \\ 1\;0\;1\;0 \ldots 0\;1\;0 \end{pmatrix} \qquad u_4 = \begin{pmatrix} 0\;1\;0\;1 \ldots 1\;0\;1 \\ 0\;0\;0\;0\;0 \ldots 0\;0\;0\;0 \\ 1\;0\;1\;0\;1 \ldots 1\;0\;1\;0 \\ \vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots \\ 0\;0\;0\;0\;0 \ldots 0\;0\;0\;0 \\ 1\;0\;1\;0\;1 \ldots 1\;0\;1\;0 \\ 0\;0\;0\;0 \ldots 0\;0\;0 \end{pmatrix}$$

Fig. 3. Four tables showing strong ambiguities

2.4 Outlier Removal

In this section we discuss possible approaches to outliers removal. During reconstruction, the most unstable parts are the borders of the image. At the borders, continuity of the function ends. This pushes the pixels at the borders away from their proper values. An attempt is made, in this paper, using a statistical approach to remove these outliers. There are many efficient algorithms for *outlier* *removal* [1], but they require a large sample of data to work correctly. We apply here a mask (i.e. a 3x3 pixels shifted over Ω) to identify potential outliers.

The *first approach* (called also *Algorithm 1*) used for outlier removal is based on Chauvenet's criterion [2]. This criterion states that any data that differ by more than two standard deviations from the mean of the data set is an outlier. The data set is created from the 9 closest non-border pixels to test one. Once an outlier is detected, it is replaced with the mean of the data set. The important feature of this test is that potential outliers are not considered as part of the sample (because if it is an outlier then it will disrupt the value of the mean). This test is done for all border pixels. Such approach unfortunately is not sufficient enough on its own (see Ex. 1).

The *second approach* (called *Algorithm 2*) is to modify 2D Leap Frog. The biggest problem with outliers was the starting edge of the image (the same for every iteration). At this point in the frame there are two borders that are unknown, and two borders that are not yet optimized. Later along this edge we have one unknown border, one optimized and two not optimized. This creates the most difficult local optimization problem in whole Ω. To remove this problem we decided that our implementation should start from different points for proceeding iterations. The change is done as follows:

1. The first iteration starts from left upper corner and proceed as described in previous paragraph.
2. Next start from right upper corner and proceed down from starting point.
3. Next start from right bottom corner and proceed left from starting point.
4. Next start from left bottom corner and proceed up from starting point.

3 Examples

This section demonstrates the difference between proposed outlier removal algorithms (i.e Algorithm 1 and 2). The last test shows also the performance of 2D Leap Frog Algorithm with respect to the size of frame. Newton's method is used here for each snapshot optimization. The Gaussian noise added to surface pictures is the same in all tests (i.e. with mean equal to 0 and deviation equal to 0.8).

3.1 Example 1

(i) The first surface S_1 is described by the graph of the function u_1 (see Fig. 4):

$$u_1(x,y) = \frac{(\cos x + \cos y)^3}{8000},$$

defined over the domain $[0,10] \times [0,10]$. The light-source directions used here are $p = (1,2,3), q = (1,8,5)$ and $r = (4,2,4)$. Gaussian noise is generated with a mean equal to 0 and deviation equal to 0.2 and added to u_1 to obtain

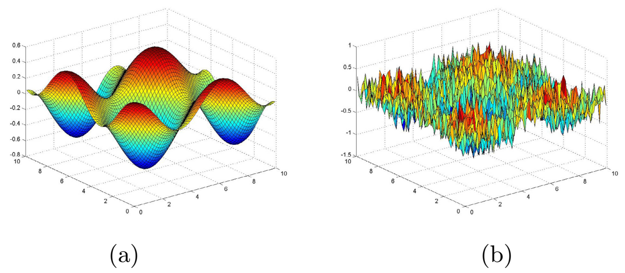


Fig. 4. (a) The ideal surface $S_1 = graph(u_1)$ and (b) the initial guess

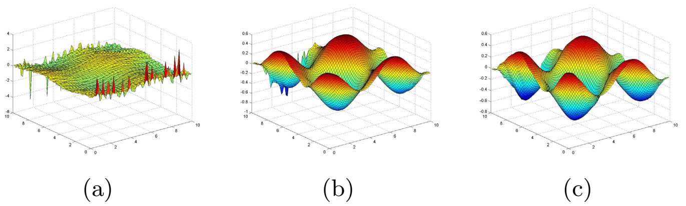


Fig. 5. Results for u_1: (a) 2D Leap Frog without outlier removal (b) Algorithm 1 - with statistical approach and (c) Algorithm 1 and 2 with both approaches

initial guess. Therefore the noise is significant as compared to the base surface plot (see Fig. 4), having values within the range of -0.6 to 0.6.
(ii) The second surface S_2 is described by the graph of the function u_2 (see Fig. 6):

$$u_2(x, y) = \cos(5 * x) + \sin(5 * y),$$

defined over the domain $[-0.5, 0.5] \times [-0.5, 0.5]$. The light-source directions used here are $p = (2, 2, 3), q = (1, 3, 2)$ and $r = (6, 2, 4)$. Gaussian noise is generated with a mean equal to 0 and deviation equal to 0.5 (see Fig. 6) and added to u_2 to obtain initial guess. The base surface plot have values is within the range of -0.2 to 0.2.

From those tests we can clearly see that our algorithm performs very good in terrain like surfaces. Also those tests shows perfectly all problems with outliers in surface reconstruction. In first case (see Fig. 5 and 7) two types of outliers were observed, ones that are on the edges of the reconstructed surface (type 2) and ones that are anywhere else (type 1). In outcome from second implementation (having outlier removal Algorithm 1 - see Fig. 5 and 7) we observe that we removed most of outliers from borders (only the strongest remain). This is because algorithm, using statistical approach, to work properly needs mean and standard deviation of the test area. Those values are taken for small area, and thus the strong outliers can disrupt them in such a manner that it will be possible to form them within the scope of accepted values. In the third approach i.e. using Algorithms 1 and 2 (see Fig. 5 and 7) we can clearly see that no big outliers are created during reconstruction process. Hence combination of Algorithms 1 and 2 for outlier removal succeeds.

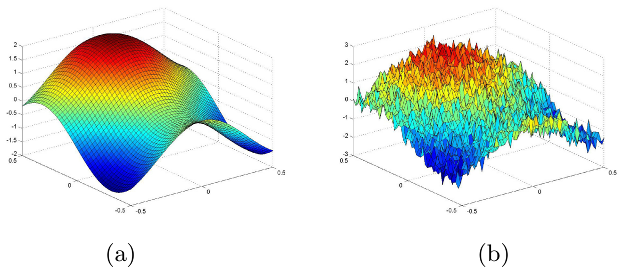


Fig. 6. (a) The ideal surface $S_1 = graph(u_2)$ and (b) the initial guess

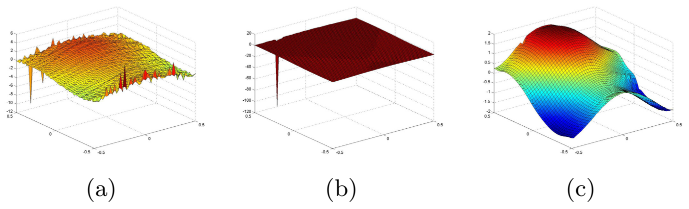


Fig. 7. Results for u_2: (a) 2D Leap Frog without outlier removal (b) Algorithm 1 - with statistical approach and (c) Algorithm 1 and 2 with both approaches

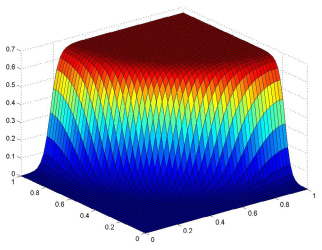
3.2 Example 2

The surface S_3 chosen for this test is described by the graph of the function u_3:

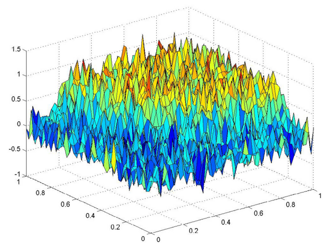
$$u_3(x, y) = \frac{1}{\left(1 - tanh\left(\frac{25}{6(4-6x+3x^2-6y^2+3y^2)}\right)\right)},$$

defined over the domain $[0, 2] \times [0, 2]$. The light-source directions used here are $p = (2, 4, 3), q = (3, 3, 2)$ and $r = (6, 2, 1)$.Gaussian noise is generated with a mean equal to 0 and deviation equal to 0.2 (see Fig. 8) and added to u_3 to obtain initial guess. The base surface plot have values is within the range of 0 to 0.8.

Table 1 shows that 6x6 is the optimum grid size for this implementation. This size provides the ideal ratio of free variables and the number of grids required to cover the image. The algorithm is run from the smallest to the largest possible grid sizes (5x5 to 8x8) for the implementation. At 9x9, containing 81 pixels, Matlab failed to run the Newton's Algorithm because there were too many unknowns in the function to minimize (about 50 unlocked variables). The next criterion is the quality of the reconstructed surface. Table 1 shows that smaller grid sizes produce reduced energies, therefore better surface reconstructions. This is because smaller functions are used to search for the local area minima. Thus the energy does not stabilize at a local minimum due to intensive grid overlaps. With bigger functions there is less change per iteration (as the whole picture is covered by fewer grids); therefore functions tend to end in its local minima, and do not change significantly in consecutive iterations.



(a)



(b)

Fig. 8. (a) The ideal surface $S_1 = graph(u_3)$ and (b) the initial guess

Table 1. Time efficiency for different grid sizes (given in minutes)

	Time needed	Iterations needed	Time per iteration	Finishing energy
5 by 5	14.5	16	0.9	0.015174
6 by 6	12.1	10	1.21	0.019644
7 by 7	17.6	14	1.25	0.093630
8 by 8	20.3	18	1.72	0.132740

4 Conclusions

It should be emphasized that 2D Leap Frog Algorithm is a very *flexible and universal tool* that can be used not only in Photometric Stereo. Clearly this computational method (with or without our modifications) is applicable to any optimization problem which in particular suffers from a large number of free variables. We close this paper with the following observations:

– We tested Leap Frog under the assumption that the Lambertian model is satisfied, and that the character of the Gaussian noise is preserved. The first simplification assures that the examined reflectance is modeled by formula (1) which is well approximated by its discrete analogue (3) (at least for image(s) with high resolution).

  In addition minimizing the cost function (3) addresses the principle of maximum likelihood (see [11]) applied to Gaussian noise added at the image level (where normal random variables denotes measurements of image intensity of each pixel). Therefore under such ideal settings Leap Frog is tested. Of course, with real surfaces, where neither Lambertian model nor Gaussian noise are preserved Leap Frog can still be tested and compared in our experiments (this forms a potential further work).

– 2D Leap Frog Algorithm (combined with outliers removals: Algorithms 1 and 2) works very well in reconstructing different surfaces (in the presence of noise).

– Our our tests clearly demonstrate that most significant changes in the cost function $\mathcal{E}$ (obtained for the whole surface S) are generated during the first 4-6 iterations. After that our cost function $\mathcal{E}$ is marginally decremented. However, it is necessary for an algorithm in question to work further because during those consecutive iterations the biggest surface outliers are removed. Unfortunately, due to space limit we were not able to include this in our tests.
– The 2D Leap Frog Algorithm outputs very good results if equipped with close enough initial guesses. If an initial guess is too far from an ideal solution to the continuous Shape from Shading then the algorithm falls within a wrong potential well of our cost function.
– According to the time criterion the best grid size for this implementation is 6 by 6 pixels snapshot for local area optimization. This grid has best ratio between the amount of grids needed to cover the whole picture with the amount of variables that are needed to be optimized.
– This technique works obviously for an arbitrary number of images. However the more images are taken the smaller Ω becomes. Also, with more images, the computational cost increases. On the other hand more images imposes tighter conditions on u. This should improve the quality of the reconstruction process.
– Possible future work is to implement 2D Leap Frog algorithm that takes into consideration shading on surface pictures (cases when $-1 \leq \cos\Theta \leq 0$ are excluded).

References

1. Atanassov, R., Bose, P., Couture, M.: Algorithms for optimal outlier removal. School of Computer Science, Carleton University, Ottawa
2. Chauvenet, W.: A Manual of Spherical and Practical Astronomy, 5th edn. Lippincott, Philadelphia (1960)
3. Horn, B.: Robot Vision. McGraw-Hill, New York (1986)
4. Horn, B., Brooks, M.: Shape from Shading. The MIT Press (1989)
5. Maruya, M., Nemoto, K., Takashima, Y.: Texture based 3D shape reconstruction from multiple stereo images. In: Proceedings of 11th IAPR International Pattern Recognition Conference A: Computer Vision and Applications, vol. I, pp. 137–140 (1992)
6. Noakes, L., Kozera, R.: A 2D Leap Frog algorithm for optimal surface reconstruction. In: Proc. SPIE 1999, Vision Geometry VII–3811, pp. 352–364 (1999)
7. Noakes, L., Kozera, R.: Denoising Images: Non-linear Leap-Frog for Shape and Light-Source Recovery. In: Asano, T., Klette, R., Ronse, C. (eds.) Geometry, Morphology, and Computational Imaging. LNCS, vol. 2616, pp. 419–436. Springer, Heidelberg (2003)
8. Kozera, R.: Existence and uniqueness in photometric stereo. Applied Mathematics and Computation 44(1), 1–104 (1991)
9. Noakes, L., Kozera, R.: Nonlinearities and noise reduction in 3-Source Photometric Stereo. Journal of Mathematical Imaging and Vision 18(II), 119–127 (2003)
10. Noakes, L., Kozera, R., Klette, R.: The Lawn-Mowing Algorithm for noisy gradient vector. In: Proc. SPIE 1999, Vision Geometry VIII-3811, pp. 305–316 (1999)
11. Zubrzycki, S.: Lectures in Probability Theory and Mathematical Statistics. American Elsevier Pub., New York (1972)